DSSynth: An Automated Digital Controller Synthesis Tool for Physical Plants

Alessandro Abate[1], Iury Bessa[2], Dario Cattaruzza[1], Lennon Chaves[2], Lucas Cordeiro[1,2], Cristina David[1], Pascal Kesseli[1], Daniel Kroening[1], and Elizabeth Polgreen[1]
[1]University of Oxford, Oxford, United Kingdom
[2]Federal University of Amazonas, Manaus, Brazil

***Abstract*—We present an automated MATLAB Toolbox, named *DSSynth* (Digital-System Synthesizer), to synthesize sound digital controllers for physical plants that are represented as linear time-invariant systems with single input and output. In particular, DSSynth synthesizes digital controllers that are sound w.r.t. stability and safety specifications. DSSynth considers the complete range of approximations, including time discretization, quantization effects and finite-precision arithmetic (and its rounding errors). We demonstrate the practical value of this toolbox by automatically synthesizing stable and safe controllers for intricate physical plant models from the digital control literature. The resulting toolbox enables the application of program synthesis to real-world control engineering problems. A demonstration can be found at https://youtu.be/_hLQslRcee8.**

***Index Terms*—Formal Synthesis; Digital Control Systems; MATLAB Toolbox; Finite-Word Length; Verification**

I. Introduction

Control theory targets the construction of reliable systems by providing mathematical guarantees about the stability and the performance of closed-loop dynamics, where outputs of a discrete plant $G(z)$ are fed back and compared to a reference signal towards which a controller $C(z)$ should steer [**?**]. Fig. 1 depicts a typical closed-loop digital control system.

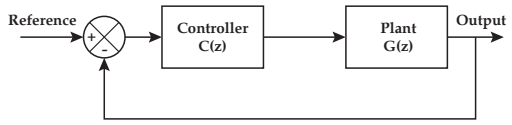


Fig. 1: Typical closed-loop control system

There are various ways to model control systems. Two common ways to represent the dynamics of such systems are transfer functions and state-space equations (difference or differential equations). In this paper, we consider systems with a single input and a single output (SISO), given either as transfer functions or state-space linear time-invariant (LTI) models. LTI models are a common sub-class, and are frequently used in cyber-physical systems (e.g., in robotics, smart grids, power plants and avionics). A discretized plant or a digital controller is, in general, given as follows when using the transfer function representation:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + ... + b_M z^{-M}}{a_0 + a_1 z^{-1} + ... + a_N z^{-N}}, \tag{1}$$

In this representation, z^{-1} is called the backward-shift operator; $A(z)$ and $B(z)$ are the denominator and numerator polynomials; and N and M are the order of the denominator and numerator polynomials, respectively.

In state-space form, the behavior of a system is represented via a state evolution equation $x(n+1)$ and an instantaneous output equation $y(n)$, as follows:

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) \\ y(n) &= Cx(n) + Du(n), \end{aligned} \tag{2}$$

where A, B, C and D are matrices that fully specify the model.

There is a rich body of literature on automating the synthesis of controllers for LTI systems [**?**], [**?**], [**?**]. However, there are many challenges when digital control is used to implement the controller, mainly due to effects of finite-word length (FWL), which leads to truncation or round-off errors [**?**], [**?**], and time discretization and quantization noise, which are typically introduced by Analogue-to-Digital (ADC) and Digital-to-Analogue (DAC) conversion [**?**].

We have recently proposed a new method to synthesize digital control systems for LTI systems, which we have named *DSSynth* (Digital-System Synthesizer) [**?**], [**?**]. The algorithm implements Counter-Example Guided Inductive Synthesis (CEGIS) [**?**]. CEGIS is used to realize a program synthesis engine that is able to generate sound digital controllers for highly non-trivial system specifications with a very high degree of automation. The program synthesis engine is given a specification and subsequently produces a sequence of candidate programs from a template. The candidate programs are iteratively refined to eventually satisfy the specification. Modern synthesis engines such as the one used here combine automated testing, genetic algorithms and SMT-based reasoning [**?**], [**?**]. Using the CEGIS-based approach, DSSynth synthesizes stable digital controllers for a given physical plant represented by either transfer functions or state-space equations.

There are existing toolboxes for MATLAB that provide functions and scripts to support the design and implementation of digital controllers [**?**]. In particular, there is a MATLAB Toolbox named *DSVerifier* [**?**], [**?**], [**?**], which automatically detects specific errors in a given digital system design (e.g., errors related to stability, limit cycle and overflow) using symbolic model checking based on Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) solvers. The Pessoa [**?**] toolbox performs synthesis of correct-by-design digital controllers. Pessoa is based on approximate

bisimulation, which allows control engineers to replace the differential equations (describing the physical plant) by a finite-state machine. However, Pessoa does not take advantage of recent advances in bit-accurate verification of programs to synthesize controllers w.r.t. FWL effects. MATLAB offers a tool named *Control System Tuner* [**?**] for automatically designing controllers based on optimization or graphical analysis. However, the Control System Tuner is unable to consider FWL effects during controller synthesis. To the best of our knowledge, there is no MATLAB toolbox able to perform digital controller synthesis that considers the complete range of approximations, including time discretization, quantization effects, and finite-precision arithmetic and its rounding errors.

The present tool paper addresses this limitation and describes a MATLAB toolbox for DSSynth. MATLAB is very commonly used for modeling physical plants. Thus, the integration into MATLAB enables rapid synthesis of a stable and safe digital controller for such plants without switching the environment. Experimental results show that the DSSynth Toolbox is able to efficiently synthesize stable and safe controllers for a set of intricate benchmarks taken from the literature: the median runtime for our benchmarks set is $124\,\mathrm{s}$, and the average synthesis time amounts to $35.5\,\mathrm{s}$ for closed-loop control systems represented by state-space equations, while for transfer functions, the average synthesis time amounts to $123.6\,\mathrm{s}$.

II. Counterexample-Guided Inductive Synthesis for Control Systems

Our synthesizer iterates between two main phases, an inductive synthesis phase, SYNTHESISE, and a validation phase, VERIFY. The two phases interact via a finite set of test vectors, which is updated incrementally. Given a stability and safety specification, the inductive synthesis procedure tries to find a candidate solution satisfying the specification for the given set of test inputs. If the synthesis phase succeeds in finding a witness, this witness is a candidate solution to the full synthesis formula. We pass this candidate solution to the validation phase, which checks whether it is a proper solution (i.e., satisfies the specification for all possible inputs). If this is the case, then the algorithm terminates. Otherwise, additional information is provided to the inductive synthesis phase in the form of a new counterexample, C-ex, which is added to the set of test inputs, and the loop iterates again.

We use two different instantiations of the synthesiser. The two approaches differ in the manner in which they check the safety specification. The first approach, presented in Fig. 2, starts by devising a digital controller that stabilizes the model while remaining safe for a pre-selected time horizon (k) and a single initial state; then, it employs a multi-staged verification process as follows: (i) The first verification stage (SAFETY) checks that the candidate solution, which we synthesized to be safe for at least one initial state, is safe for *all* possible initial states, i.e., does not reach an unsafe state within k steps. (ii) The second verification stage (PRECISION) restores soundness with respect to the plant precision by using interval arithmetic [**?**] to validate the operations performed by the previous stage. (iii) The third verification stage (COMPLETE)

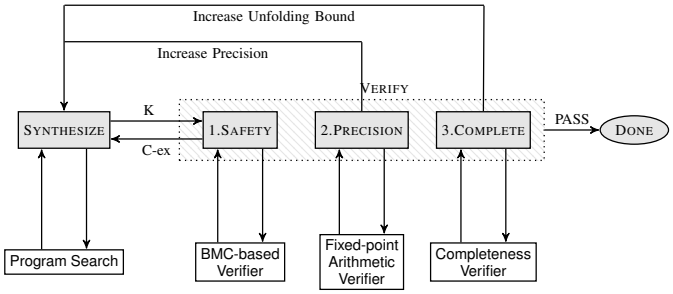


Fig. 2: CEGIS with multi-staged verification

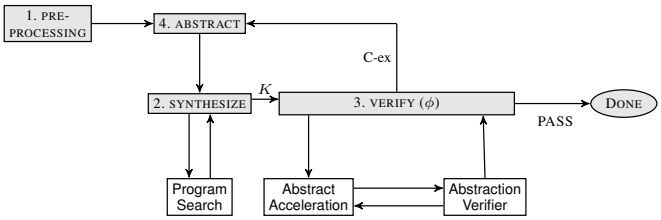


Fig. 3: Abstraction-based CEGIS

checks that the current k is large enough to ensure safety for any $k'>k$. Here, we compute the completeness threshold $\overline{k}$ for the current candidate controller K [**?**], i.e., the number of iterations required to sufficiently unwind the closed-loop state-space model such that the boundaries are not violated for any larger number of iterations, and check that $k\geq\overline{k}$.

The second approach (illustrated in Fig. 3) employs *abstract acceleration* [**?**] to simultaneously evaluate all possible progressions of the model. This eliminates the need for a completeness threshold, since the time horizon considered in abstract acceleration is infinite. An additional preprocessing phase is used to compute a set of initial bounds on K based on input constraints. Note that these bounds will be used by the SYNTHESIZE phase to reduce the size of the solution space. Since this model is based on abstraction, we initially perform the synthesis over a basic model that abstracts away the complexity of the specification and is only refined to a more complex model when a counterexample is found. The use of this Counterexample-Guided Abstraction-Refinement results in much faster synthesis and verification since it is applied to both phases, allowing for faster results whilst maintaining soundness for an infinite time horizon.

III. Synthesizing Digital Controllers with the DSSynth Toolbox

A. DSSynth Toolbox Architecture

The proposed synthesis methodology for closed-loop digital control systems is based on the DSSynth tool [**?**], [**?**], which can be split into two main stages as follows: manual and automated steps, as illustrated in Fig. 4.

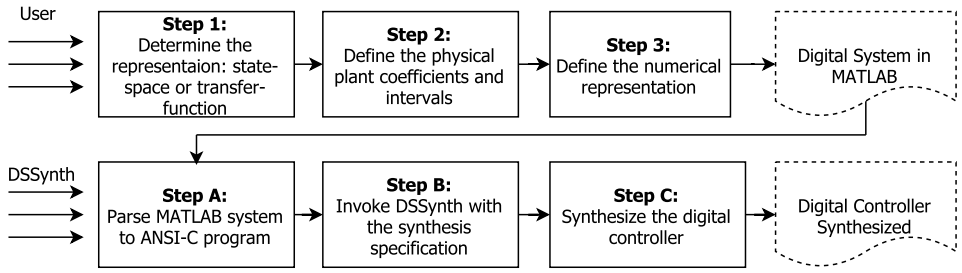


Fig. 4: Phases of the controller synthesis

In Step 1, the user selects the digital controller representation, which can be a transfer function or a state-space model. In Step 2, the physical plant (in the form of Equation (1) or (2), depending on representation) must be given [**?**]. Finally, in Step 3, the numerical representation for the digital controller implementation must be set by the user, i.e., the FWL format that defines the number of bits of the integer and fractional parts as well as the dynamic range inputs. The input provided by the user is a specification of the physical plant.

After that, the automated synthesis process starts with Step A, where the DSSynth Toolbox translates the digital system specification from the MATLAB format into an ANSI-C program. In Step B, our CEGIS engine is invoked to synthesize the digital controller w.r.t. the digital system specification. Finally, in Step C, the synthesized digital controller is produced. The output generated by the DSSynth Toolbox is the synthesized digital controller represented either in transfer function or state-space equation. The synthesis is considered to be *successful* if a digital controller is correctly synthesized w.r.t. FWL effects; otherwise, if any parameter is incorrectly defined in previous steps, or if there is no solution (no controller) for the respective physical plant (given the implementation requirements), then the synthesis is considered to have *failed*.

Our CEGIS engine is implemented as an integrated module within the C bounded model checker (CBMC) [**?**]. CBMC transforms the ANSI-C representation of our closed-loop control system model into its internal format. We instrument this IR for each synthesis or verification scenario accordingly, and use CBMC as an oracle to answer our queries. CBMC itself relies on an underlying SAT or SMT solver to solve verification conditions. We model FWL effects, time discretization and the presence of quantization noise explicitly using CBMC's nondeterminism API (e.g., `nondet`, `CPROVER_assume` intrinsic functions). Our module is included in CBMC 5.8 and is available for download.[1] We further created a VirtualBox OVA image with our tool pre-installed, including our MATLAB-generated benchmarks and benchmark shell script with instructions to run all benchmarks and reproduce our experiments.[2] Our toolbox is available online.[3]

The entire implementation consists of 400 lines of MATLAB scripts for the UI module, 585 and 800 lines of C code in the naive and abstract accelerator back-ends, respectively (cf. Section II), 4000 lines of C code for utility functions such as interval and fixed-point arithmetic, as well as 1000 lines of C++ code in the control synthesis module in CEGIS CBMC. We evaluate DSSynth Toolbox using 18 SISO control system benchmarks. Additionally, documentation and videos about the DSSynth Toolbox are available at www.cprover.org/DSSynth/dssynth-toolbox-1.0.0.zip.

[1]https://github.com/diffblue/cbmc/archive/cbmc-5.8.zip
[2]www.cprover.org/DSSynth/controller-synthesis-cav-2017.tar.gz
[3]https://github.com/ssvlab/dsverifier/tree/master/toolbox-dssynth

B. DSSynth Toolbox Procedures

The DSSynth Toolbox performs the following automated steps to synthesize a digital controller for the given physical plant:

1) **Setup**: obtains the physical plant, fixed-point format and dynamic input ranges, and translates them to a specific structure in MATLAB.
2) **Parse**: obtains the digital system specification and translates it into an ANSI-C program.
3) **Execution**: obtains the ANSI-C program from the previous (parse) step and calls our CEGIS engine as a back-end program synthesis tool to perform the automated synthesis.
4) **Extraction**: obtains the ".log" file that is generated after the synthesis phase and then checks the synthesized digital controller.
5) **Report**: obtains the digital controller from the previous (extraction) step and translates it into MATLAB, for presentation to the user.

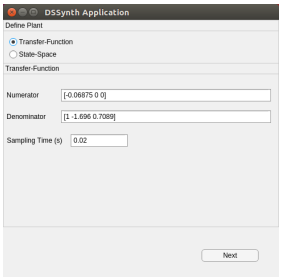
C. DSSynth Toolbox Usage

1) Command Line: Users must provide a digital system described as a MATLAB model using a `tf` (for transfer function) or an `ss` (for state-space model) command (Step 1 of Fig. 4). The DSSynth Toolbox is called via the command line in MATLAB as

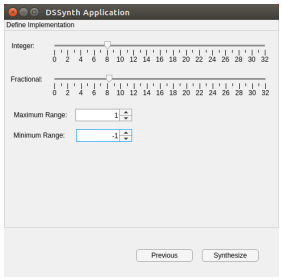
```
synthesize(plant, intBits, fracBits, maxR, minR)
```

where `plant` is the physical plant in state-space or transfer function representation, `intBits` is the integer part, `fracBits` is the fractional part, `maxR` and `minR` are the maximum and minimum dynamic range, respectively (Steps 2 and 3 of Fig. 4). After executing the `synthesize` command, the DSSynth Toolbox provides the synthesized digital controller.

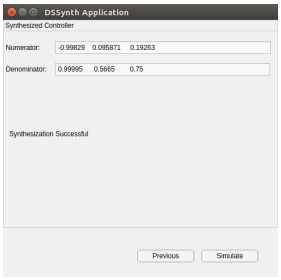
2) MATLAB Application: A graphical user interface application was developed (illustrated in Fig. 5) to support digital controller synthesis in MATLAB. Its aim is to improve usability and, consequently, to attract more engineers. Users provide the following inputs for digital controller synthesis within the UI: physical plant specification, fixed-point representation and the range of the inputs.



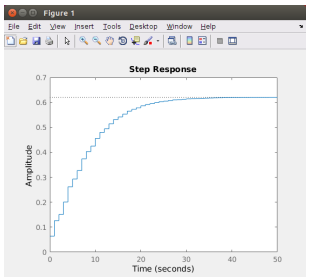
(a) Definition of the system representation and the physical plant



(b) Definition of implementation aspects and input ranges



(c) Digital controller synthesized by DSSynth



(d) Step response for the synthesized digital controller

Fig. 5: DSSynth GUI for controller synthesis using transfer function representation

D. Illustrative Example

Fig. 6 illustrates each step of the synthesis process for a physical plant described in Eq. (3), which represents an unmanned aerial vehicle (UAV) quadcopter system [**?**]. This physical plant is expressed by a transfer function $G(z)$, where $B(z)$ and $A(z)$ represent the numerator and denominator, respectively, as

$$G(z) = \frac{B(z)}{A(z)} = \frac{-0.06875z^2}{z^2 - 1.696z + 0.7089}. \quad (3)$$

```
>> num = [-0.06875 0 0];
>> den = [1.0000 -1.696 0.7089];
>> system = tf(num,den,0.002);
>> y = synthesize(system,8,8,1,-1);
>> SYNTHESIS SUCCESSFUL
>> y =
>>  -0.9983z^2 + 0.09587z + 0.1926
>>  --------------------------------
>>       z^2 + 0.5665z + 0.75
```

Fig. 6: Synthesis of a digital controller for the physical plant defined in Eq. (3) with fixed-point format $\langle 8, 8 \rangle$

In Fig. 6, "num" represents $B(z)$ and "den" represents $A(z)$, while y represents the synthesized stable digital controller, which is defined by Eq. (4).

$$C(z) = \frac{-0.9983^2 + 0.09587z + 0.1926}{z^2 + 0.5665z + 0.75}. \quad (4)$$

A digital system is stable iff all of its poles are inside the z-plane unity circle, i.e., the poles must have a modulus less than one [**?**]. In order to compute the poles for the digital system represented by Equations (3) and (4), the user can invoke the `feedback` and `series` commands in MATLAB [**?**] to obtain a general equation regarding the transfer function representation, as illustrated in Fig. 7.

```
>> num = [-0.99832 0.09587 0.1926];
>> den = [1 0.5665 0.75];
>> controller = tf(num,den,0.002);
>> num = [-0.06875 0 0];
>> den = [1.0000 -1.696 0.7089];
>> plant = tf(num,den,0.002);
>> sys = feedback(series(controller, plant),1)
>> sys =
>>         0.06863z^4 - 0.006591z^3 - 0.01324z^2
>> ------------------------------------------------------
>> 1.069z^4 - 1.136z^3 + 0.4849z^2 - 0.8704z + 0.5317
```

Fig. 7: General equation for the closed-loop control system using the series configuration as illustrated in Fig. 1

The general equation that represents the closed-loop control system, using the series configuration as illustrated in Fig. 1, is described by Eq. 5 as follows:

$$\frac{N(z)}{D(z)} = \frac{0.06863z^4 - 0.006591z^3 - 0.01324z^2}{1.069z^4 - 1.136z^3 + 0.4849z^2 - 0.8704z + 0.5317}. \quad (5)$$

For the stability check, if any root of the denominator polynomial $D(z)$ in Eq. 5 has modulus equal or greater than one, then the system is unstable; otherwise, it is stable. Indeed, the roots computed by the function `roots` in MATLAB [**?**] result in the following poles as illustrated in Fig. 8, which means that DSSynth has synthesized a stable digital controller.

```
>> den = [1.069 -1.136 0.4849 -0.8704 0.5317]
>> r = roots(den)
>> r =
>>   -0.2912 + 0.8061i
>>   -0.2912 - 0.8061i
>>    0.8225 + 0.0219i
>>    0.8225 - 0.0219i
```

Fig. 8: Roots of the polynomial $D(z)$ given by Eq. 5

Simulating the digital system represented by Equations (3) and (4) to validate and reproduce the system stability, the user can obtain the step response using the command `dstep` in MATLAB, and then observe that the plotted graph indicates a stable digital system (Fig. 9).

IV. DSSynth Toolbox Experimental Evaluation

A. Benchmark Description

Our evaluation consists of 18 SISO control system benchmarks extracted from the literature [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], which include physical

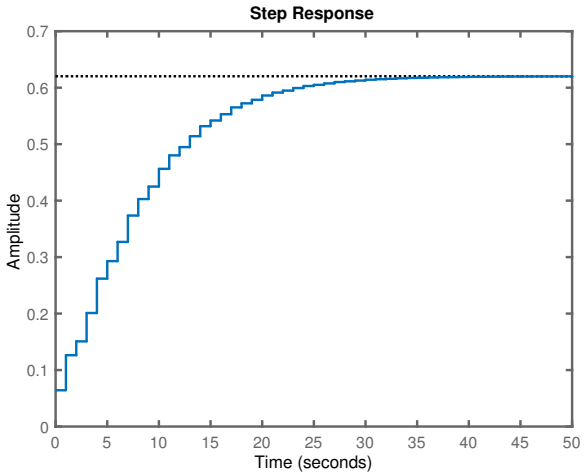


Fig. 9: Step response for Eq. (3) describing a stable system for the UAV quadcopter system

plants for an unmanned aerial vehicle (UAV), a cruise control system, a spring-mass damper, a satellite, a direct current (DC) servo motor, a helicopter, the inverted pendulum, a pendulum, magnetic suspension, a magnetized pointer, 1/4 car suspension, a computer tape driver, a flexible beam, a guidance system, a US Coast Guard cutter, a continuous stirred tank reactor, a voltage regulator, and an acrobot control system.

B. Experimental Setup

For all evaluated benchmarks, the signal input range is between -1 and 1. The implementation features used for digital controller are: 8 bits for the integer part (including the sign bit) and 8 bits for the fractional part. We have run the DSSynth Toolbox using CEGIS with multi-staged verification (cf. Section II). All experiments with DSSynth Toolbox v1.0.0 were conducted on an otherwise idle Intel Core i7 $-$ 2600 3.40 GHz processor, with 24 GB of RAM, running Ubuntu 64-bit.

C. Objectives

The main objectives of our experimental evaluation is to (1) evaluate the DSSynth Toolbox performance to produce stable and safe controllers, and also (2) confirm the stability and safety of the synthesized digital controllers outside of our model using MATLAB.

D. Results

For our experiments, we observed that the DSSynth Toolbox correctly synthesizes stable and safe digital controllers for our set of benchmarks represented by transfer functions or state-space equations. We employed a diverse set of real-world benchmarks extracted from the control literature. However, note that this set of benchmarks is still limited within the scope of this paper, and the performance may not generalize to other benchmarks. The median runtime for our benchmarks is $124\,$s, and the average synthesis time amounts to $35.5\,$s for closed-loop control systems represented by state-space equations, while for transfer functions, the average synthesis time amounts to $123.6\,$s.

Additionally, we are able to reproduce the stability and safety for all evaluated control system benchmarks. As an example, we have reproduced the stability property in the synthesized controller for a DC motor plant described in Eq. (6), simulating the step response in MATLAB. Fig. 10 shows that the digital controller synthesized by the DSSynth Toolbox stabilizes the closed-loop control system using the series connection (illustrated in Fig. 1) for the DC motor plant. We have applied this procedure to all synthesized controllers to validate our experimental results.

$$C(z) = \frac{-0.3466796875z + 0.015625}{-0.5z^2 + 0.19921875z}. \tag{6}$$

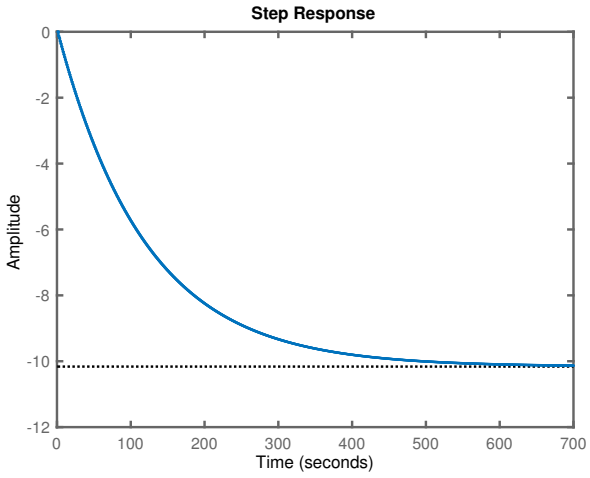


Fig. 10: Step response for the Eq. (6) describing a stable system for the DC motor plant

Regarding the state-space representation, we have reproduced the stability property for the closed-loop control system of a pendulum represented by Eq. (7).

$$\begin{gathered} A = \begin{bmatrix} -1.999909737361046 & -1.000000000000000 \\ 1 & 0 \end{bmatrix}, \\ B = \begin{bmatrix} 4 \\ 0 \end{bmatrix}, \\ C = \begin{bmatrix} -1.756887232846049 & -1.7494556607764090 \end{bmatrix}, \\ D = \begin{bmatrix} 9.8 \end{bmatrix}. \end{gathered} \tag{7}$$

For this particular closed-loop control system, the feedback matrix (see Eq. (8)) generated by the DSSynth Toolbox also leads to a stable system, as illustrated in Fig. 11.

$$K = \begin{bmatrix} -0.5898 & -0.1914 \end{bmatrix}. \tag{8}$$

The synthesized controllers are also confirmed to be safe outside of our model representation using MATLAB, which means that our results are usable for control engineers. Given that the closed loop dynamics are convergent, the controller's safety can be checked by computing the initial state's infinity norm, i.e., the product between the l_∞ norm of the initial states

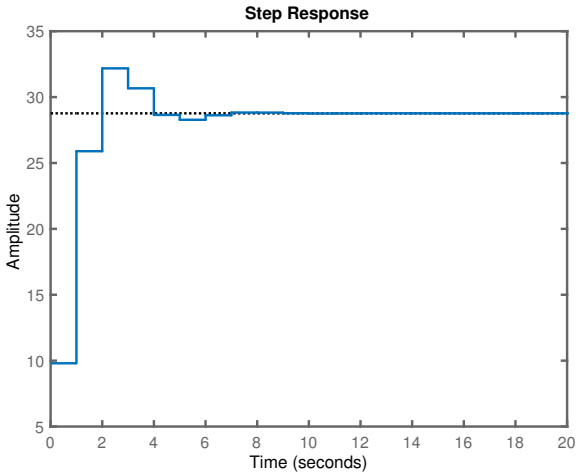


Fig. 11: Step response for the Eq. (7) describing a stable system for the pendulum plant

($||x_0||_\infty = \max x_0$) and the l_1 norm of ($||\bar{A}||_1 = \max_j \sum_{i=1}^{m} |\bar{a}_{ij}|$, where $\bar{A} = A - BK$). This is a conservative safety check, which is based on a sufficient condition [**?**]. An alternative (but not conservative) approach is to unfold the system dynamics up to a given depth, with the initial state equal to its maximum value, compute the system input and output via the feedback matrix, and then check the state bounds.

V. Conclusions

The DSSynth Toolbox can automatically synthesize stable and safe digital controllers for dynamic physical plants, represented in MATLAB either as a transfer function or a state-space equation. The DSSynth Toolbox is the first fully automated synthesis tool that is algorithmically and numerically sound, considering various error sources in the implementation of the digital control algorithm and in the computational modeling of plant dynamics [**?**], [**?**]. The DSSynth Toolbox presents a practical application of CEGIS-based program synthesis. Our results in control engineering encourage the application of this technique in other fields such as program repair or superoptimization. Additionally, given the current literature in controller synthesis, there is no other MATLAB toolbox for synthesizing stable and safe digital controllers for physical plants that considers all aspects of their digital implementation. As future work, the DSSynth Toolbox will perform synthesis considering performance requirements and will be combined with the DSVerifier/DSValidator Toolboxes [**?**], [**?**] to verify/-validate other controller properties. We will also pursue the application of CEGIS to further software engineering problems.